

CODE GENERATION

When should we use it?
When should we avoid it?

THE PROBLEM

- ◆ Many applications that use databases involve a large amount of repetitious boilerplate code, which users don't want to maintain and which they'd prefer not to understand. To avoid this, we have made use of code generators.
- ◆ We also use code generators for other purposes...

EXAMPLES WE USE

- ◆ D0 and CDF calibration database access
 - ✦ CDF writes a Java specification of database tables and rows; code generator executes this to produce C++ classes the users see and the back-end code which interacts with a variety of databases.
 - ✦ D0 does a query to Oracle to generate Python, which is parsed to generate C++ structs and CORBA IDL for the client, and Python access code for the server.
- ◆ rootcint and d0cint for persistency
- ◆ rootcint for dictionary for interactive use
- ◆ Qt: GUI generator and MOC

MORE EXAMPLES WE USE

◆ Java

- ◆ GUI builders
- ◆ RAD tools with servlet generators, beans generators, *etc.*

◆ SWIG and boost.python

- ◆ Wrap existing C or C++ for use in another language

◆ CORBA IDL

◆ flex/bison generated parsers

EXAMPLES WE DO NOT USE

- ◆ Rational Rose, or any other UML --> C++ generation
- ◆ C++ RAD tools with application builders
 - ◆ Why do we use them for Java, but not C++ or Python?

QUESTIONS WE SHOULD ADDRESS

- ◆ What classes of problems do code generators solve well? What features should we look for to know we should rule out code generation?
- ◆ In a pure C++ environment, for what sort of problem would code generation be clearly superior to use of templates?
- ◆ How can we design or choose code generation systems to avoid the problems listed?
- ◆ What additional benefits could we gain, that we are not now enjoying?

WAYS TO CLASSIFY TOOLS

- ◆ Input language
- ◆ Output language
- ◆ Developer interaction with output
- ◆ User interaction with output
- ◆ Level of abstraction of output

DESIGN PHILOSOPHIES

- ◆ Token merging into a template
 - ✦ This is how the CDF code generation works
 - ✦ Jakarta struts does this, for generating dynamic web content
- ◆ Code generator with built-in mapping from input specification to output code.
- ◆ Interface Definition Language (IDL)
 - ✦ CORBA
 - ✦ SWIG

MORE DESIGN PHILOSOPHIES

- ◆ General purpose language as input
 - ◆ boost.python
- ◆ Mark-up of general purpose language as input
 - ◆ rootcint, d0cint
 - ◆ Qt MOC
- ◆ Special-purpose language with embedded code segments
 - ◆ flex/bison, lex/yacc

AND MORE!

- ◆ Generation of code skeleton to be filled in by developer
 - ◆ RAD tools

FOCUSING THE DISCUSSION

- ◆ The applications in which code generation is used cover a huge range.
- ◆ We want to focus on a particular application domain: persistency, including (and most importantly) database access.

DIFFICULTIES ENCOUNTERED

- ◆ Tight coupling between database tables and client code, and everything in between
- ◆ Code bloat
- ◆ Synchronization of development for multiple back ends, *e.g.* Oracle and MySQL
- ◆ Representation and maintenance of template (boilerplate) code.
 - ◆ Having C++ code produced by C++ or python or Java
 - ◆ Comprehending the code (understanding its purpose and design)

QUESTIONS WE SHOULD ADDRESS

- ◆ What classes of problems do code generators solve well? What features should we look for to know we should rule out code generation?
- ◆ In a pure C++ environment, for what sort of problem would code generation be clearly superior to use of templates?
- ◆ How can we design or choose code generation systems to avoid the problems listed?
- ◆ What additional benefits could we gain, that we are not now enjoying?